

A NEW TWIST FOR THE SIMULATION OF HYBRID SYSTEMS USING THE TRUE JUMP METHOD

ROMAIN VELTZ

Abstract. The use of stochastic models, in effect piecewise deterministic Markov processes (PDMP), has become increasingly popular especially for the modeling of chemical reactions and cell biophysics. Yet, exact simulation methods, for the simulation of these models in evolving environments, are limited by the need to find the next jumping time at each recursion of the algorithm. Here, we report on a new general method to find this jumping time for the *True Jump Method*. It is based on an expression in terms of ordinary differential equations for which efficient numerical methods are available. As such, our new result makes it possible to study numerically stochastic models for which analytical formulas are not available thereby providing a way to approximate the state distribution for example. We conclude that the wide use of event detection schemes for the simulation of PDMPs should be strongly reconsidered. The only relevant remaining question being the efficiency of our method compared to the *Fictitious Jump Method*, question which is strongly case dependent.

Key words. PDMP, piecewise deterministic Markov process; ODE, ordinary differential equations.

1. Introduction. There is a growing number of processes (from biophysics, ecology, finance, internet traffic, queuing...) that are inaccurately modeled with ODEs. A classical example is the one of chemical reactions for which the reactants are well stirred but in such a small number that a continuous description is inappropriate. A description in term of a Markov process is more relevant in this case [Gil77].

The study of Markov models, with intrinsic noise, comes with the computation of the distribution of the states at a given time. Other methods, aimed at understanding the noisy dynamics, rely on sample paths [Arn98]. Simulation methods which provide exact sample paths are very helpful for the computation of the state distribution. There are two broad classes of simulation methods [GT13, CT97], the *Fictitious Jump Method*¹ (FJM) and the *True Jump Method* (TJM), see appendices. The last method was long known before being popularized by Gillespie [Gil77, GB00] such that it bears the name of the Doob-Gillespie algorithm.

There are several limitations to the above algorithms. The first occurs when the number of chemical reactions is large, the second occurs when there is a subpopulation with very fast kinetics compared to the other reactions and the third being if the transition rates², are time dependent (in the case of the TJM). The first two limitations lead to very long simulation times while the last one requires to accurately solve an integral equation at each iteration which can be difficult and time consuming [And07] (but see [CT97] for a FJM version).

The first two limitations can be addressed by using an approximate algorithm, for example the *tau-leaping method* [Gil01], or by using a time-scale separation [KK13] which provides a PDMP as limit of the suitably scaled Markov model. In short, the limit is composed of an ODE, averaging the fast component of the Markov model and a jump process representing the slow component of the original Markov model.

Finally, PDMPs [Dav84, Dav93] also arise in models in which one wants to couple deterministic dynamics to a chemical process, as for example in the modeling of the membrane potential of a neuron coupled with the stochastic opening/closing of the ions channels [Bre14, PTW10]. The PDMPs are composed of two components: a continuous one described by the flow of an ODE and a jump process - that affects the continuous flow and describes the stochastic component of the process. Note that this class of Markov processes encompasses the one of chemical reactions with time dependent reaction rates as a subtype.

¹also known as rejection method

²also called *propensity functions* in the case of chemical reactions

This motivates the need for the computation of *exact* sample paths of PDMPs. There are here two possibilities:

1. If the continuous flow is known analytically, then the FJM is an exact method. Unless the integral equation in the TJM can be solved analytically as well, the TJM is not *exact* in this case and should be discarded in favor of the FJM.,
2. If the continuous flow is not known analytically, none of both methods is exact. The error made in the simulation comes from the need to compute numerically the continuous flow, and in the case of the TJM, from the numerical computation of the jumping times. Hence, compared to the FJM, the TJM adds another error source. We shall see that our method addresses this last point.

The simulation of PDMPs [ACT⁺05, BdSD10, GPB08, Rie13] is reviewed in [Rie13] where the author justifies theoretically the original method of [ACT⁺05], albeit focusing only on the TJM. They propose a method based on an ODE solver with event detection [Sha03] where the event is the integral equation that needs to be solved to find the jump instants (see appendix A). Such numerical methods may not be available and suffers from two majors drawbacks. First, one may need to modify a library providing an ODE solver to implement the event detection. Given the complexity of such solvers [HW96], this may not be trivial. The second issue with the use of event detection scheme is that the time-step is *only* adapted to solve the ODE and not to locate precisely the event. This is a general word of caution concerning event detection for ODE, it is an ill-posed problem [Sha03] and may lead to excessively large numerical errors as we shall see. Finally, note that all examples in [Rie13] are non-stiff, hence being relatively easy to solve numerically. We address all these issues with one theoretical result.

Here, we report on a new simulation algorithm derived from the TJM. It is based on the assumption that a regular ODE solver is available without further capabilities like event detection. The precision of our method is exactly the one of the ODE solver [Ske86, HW96] making the simulation of PDMP nothing more than the simulation of an ODE. In particular, global error estimation [Ske86, Vis01, CP04] for ODE directly leads to global error estimation of the simulation of the PDMPs. Our new method is based on a change of time variable and recursively solving an ODE of dimension $n + 1$ where n is the dimension of the continuous flow of the PDMP. Hence, it comes with little additional cost given that one needs to simulate the continuous component of the PDMP anyway.

The only relevant remaining question is how our new method compares to the FJM when the continuous flow is not known analytically. We will come back to this question and provide a numerical example.

The paper is organized as follows. We first derive the theoretical result which is the basis of our method. Then we provide three numerical examples. In the first two examples, we show that the simulation methods based on event detection are to be reconsidered. We then provide a third example showing the effectiveness of our method compared to the FJM before drawing some conclusions.

2. Description of the PDMPs. In this paper, we consider a PDMP by following [GT13] in order to simplify the description of the process. Note that a more general definition is provided in [Dav84, Dav93] but we stick to [GT13] for simplicity. A PDMP is a jump process $(X(t))_{t \in \mathbb{R}^+}$ where the stochastic jump instants T_n are non decreasing, isolated with limit $\lim_{n \rightarrow \infty} T_n = \infty$ and the dynamics between the jumps is given by a flow of homeomorphisms

$(\phi^t)_{t \geq 0}$ on \mathbb{R}^d as follows:

$$\begin{cases} \forall t \geq 0, & X(t) = \sum_{n \geq 0} \phi^{t-T_n}(X(T_n)) \mathbf{1}_{\{T_n \leq t < T_{n+1}\}}, \\ 0 = T_0 < T_1 \leq T_2 \leq \dots \leq \infty, \\ T_n < \infty \Rightarrow T_n > T_{n-1} \quad \text{and} \quad X(T_n) \neq X(T_n-). \end{cases} \quad (2.1)$$

We assume that the continuous flow is produced by an ODE:

$$\begin{cases} \frac{d\phi^t(x)}{dt} = F(\phi_t(x)), & t \geq 0 \\ \phi_0(x) = x. \end{cases} \quad (2.2)$$

such that $x \rightarrow F(x)$ is locally Lipschitz. The jump $X(T_n)$ and the inter jump interval $T_n - T_{n-1}$ are independent. The law of $X(T_n)$ given $X(T_{n-1}) = x$ is $\Pi(x, dy)$ and the law of the inter jump interval follows from

$$P(T_n - T_{n-1} > t | X(T_{n-1}) = x) = \exp\left(-\int_0^t R_{tot}(\phi^s(x)) ds\right)$$

where the total transition rate function R_{tot} , is measurable and satisfies $R_{tot}(x) \geq 0$. We allow

$$\sup_{x \in \mathbb{R}^d} R_{tot}(x) \leq \infty$$

but R_{tot} must be locally bounded. This implies [GT13] that there is a finite number of jumps on any bounded time interval.

Remark 1. Using the usual trick, we can deal with the case where the vector field F in (2.2) is time-dependent, i.e. $F : (x, t) \rightarrow F(x, t)$, but the existence of PDMP holds for less restrictive assumptions: $t \rightarrow F(x, t)$ can be assumed continuous rather than Lipschitz as stated above. The same trick also allows to deal with the case where R_{tot} is time-dependent as well.

3. Theoretical result. A general simulation algorithm, called the True Jump Method [GT13], is provided in appendix A: it is also a general method to prove existence of PDMPs [GT13] under the conditions stated in the previous section. A serious limitation to this algorithm is the need to solve the integral equation in step 3. The limitation pertains to the case³ $F = 0$ if the total transition rate function R_{tot} is time-dependent. In general, even if the flow (ϕ^t) is known analytically, the integral equation needs to be solved numerically using Newton method or the bisection algorithm for example. Here, we propose a solution based on an ODE solver and a change of time variable.

THEOREM 3.1. *Let us make the general assumptions:*

- the flow $(\phi^t)_{t \geq 0}$ is globally defined,
- $R_{tot}(x) > 0$ for all $x \in \mathbb{R}^d$,
- $x \rightarrow R_{tot}(x)$ is locally Lipschitz on \mathbb{R}^d ,

Then, the solution $(X(T_n^-), T_n)$ of the integral problem in step 3. is given by $(y(S_n), \tau(S_n))$ where

$$\begin{cases} \dot{y}(s) = F(y(s)) / R_{tot}(y(s)) \\ \dot{\tau}(s) = 1 / R_{tot}(y(s)) \\ y(0) = X(T_{n-1}), \tau(0) = T_{n-1}. \end{cases} \quad (3.1)$$

³the continuous function is stationary in this case

Also $X(\tau(s)) = y(s)$ for $s \in [0, S_n)$.

Proof. The trick for solving $\int_{T_{n-1}}^u R_{tot}(X(s))ds = S_n$, in Algorithm 1., consists in the introduction of the variable $\tau(s) \geq T_{n-1}$ defined as

$$\int_{T_{n-1}}^{\tau(s)} R_{tot}(X(t))dt = s$$

which gives the second equation in (3.1) by differentiating with respect to s . However, we need to know $X(\tau(s))$ to solve the ODE for $\tau(s)$. Writing $y(s) \stackrel{def}{=} X(\tau(s))$ and using chain rule differentiation gives the remaining equation in (3.1). \square

The fact that the flow ϕ is globally defined (no "explosions") is a simplifying assumption of [Dav84]. The theorem states that solving the integral equation in step 3. of the Algorithm 1. amounts to integrate an ODE over a time interval $[0, S_n)$ where S_n is randomly drawn from an exponential distribution $\mathcal{E}(1)$ at each recursion of the algorithm.

Remark 2. It is easy to generalize the theorem to the case where F, R_{tot} are time-dependent. One finds

$$\begin{cases} \dot{y}(s) = F(y(s), \tau(s)) / R_{tot}(y(s), \tau(s)) \\ \dot{\tau}(s) = 1 / R_{tot}(y(s), \tau(s)) \end{cases} \quad (3.2)$$

3.1. Sampling the continuous flow. The previous method is effective for computing the jump instants and the values of the PDMP at these instants. However, if one seeks to sample the continuous part in between jumps, there is a difficulty because one only has access to X through a change of time variable τ . A cheap way around this issue is by adding a Poisson process $(N(t))_{t \in \mathbb{R}^+}$ of constant rate $\lambda > 0$ to our PDMP, thereby increasing the dimension of our system: $d \rightarrow d + 1$. This way, Theorem 1. condition $R_{tot}(x) > 0$ is always satisfied because $R_{tot}(x) \geq \lambda > 0$. This additional jump process will add jump events at rate λ , hence sampling the continuous dynamics.

3.2. Simulations. The simulation of the process $(X(t))$ consists in applying the Algorithm 1. together with the use of Theorem 1. We call this procedure the CHV method. The performance of the ODE solver strongly impacts the speed and "precision" of the simulation. It is expected that the equations (3.1) may be stiff depending on whether a lot (resp. a few) reactions are initiated in which case the factor $1 / R_{tot}$ might assume vanishing or extremely large values. Hence, we find it convenient to use the ODE solvers [Hin83] for their ability to automatically switch between stiff and non-stiff methods. Note that depending on the problem, other ODE methods [HW96] may be more appropriate than the one suggested.

3.3. Comparison with the FJM. As stated in the introduction, when the flow is known analytically, there is no point in choosing a quasi-exact method *e.g.* the True Jump Method instead of the exact FJM, unless of course, if the jump instants are known analytically.

When the flow is not known analytically, both methods FJM/CHV are quasi-exact in that they bear a numerical error coming from integrating the ODE underlying the continuous component. The question we ask is which method is the fastest given a numerical tolerance.

For convenience, we recall the two ODEs that we need to solve between the jumps, on an interval of length $S_n \sim \mathcal{E}(1)$, for each method:

$$(FJM) : \dot{x} = F(x)/\lambda \quad \text{or} \quad (CHV) : \begin{cases} \dot{y} = F(y)/R_{tot}(y) \\ \dot{\tau} = 1/R_{tot}(y) \end{cases} . \quad (3.3)$$

If $\lambda \approx R_{tot}(x)$, then (FJM) is smaller hence faster to solve. However, if $R_{tot}(x)$ varies a lot, it may be difficult to bound it effectively using a single constant λ as required in the FJM (see appendix B). Hence, the choice of the method boils down to know whether it is faster (and more accurate) to solve k times⁴ (FJM) or a single time (CHV). This depends a lot on the time stepper and the way it copes with the numerical tolerances to produce the (adaptive) time steps.

In the case of fixed time steps, one needs to see whether the cost of the larger system (CHV) compensates the cost of the generation of the k random numbers. Hence, if the FJM does not require a lot of fictitious jumps, this method is more appropriate than CHV.

Finally, if the transition rate R_{tot} is zero on an open set, then the CHV method breaks down and the FJM is more appropriate. On a practical side, the CHV method is simpler to program but the FJM assume far less regularity for R_{tot} .

4. Numerical examples. We give two examples where our new method is particularly effective compared to the standard use of event location for solving the integral equation to compute the jumping times. Then we provide an example to compare our method to the FJM.

4.1. Stiff problem with possibly infinite jumping times. The purpose of this example is to show a case where the event location in `Matlab` is actually doing a good job and compare it to our method. To make things difficult for the event location, we chose stiff dynamics between the jumps. Note that the next jumping time can be infinite in our example. We look at the following process of switching dynamics where $X(t) = (x_c(t), x_d(t)) \in \mathbb{R}^2$:

$$\begin{cases} \dot{x}_c = a(t)x_c \\ a(t) = -100 \times (2(x_d(t) \bmod 2) - 1) \end{cases} \quad (4.1)$$

starting from $x_c(0) = 1$. The (unique) transition rate is $R_{tot}(x_c, x_d) = x_c$ associated with the jump $x_d \rightarrow x_d + 1$ for which $\Pi((x_c, x_d), dy) = \delta_{(x_c, x_d+1)}(dy)$. As x_c increases, the transition rate also increases. Hence, the continuous problem is stiff because one has to make sure (numerically) that $R \geq 0$ despite the abrupt decrease of x_c . Also, the integral equation is difficult to handle numerically with high precision. The jumping time, solution of the integral equation, reads (see Theorem 3.1):

$$\begin{aligned} a &\stackrel{def}{=} -100 \times (x_d(T_{n-1}) \bmod 2 - 1) \\ T_n &= T_{n-1} + \frac{1}{a} \log(1 + aS_n/x_c(T_{n-1})), \\ X(T_n) &= X(T_{n-1}) + aS_n. \end{aligned} \quad (4.2)$$

From the formula, it appears that the next jumping time can be infinite when $1 + aS_n/x_c(T_{n-1}) < 0$ when $a < 0$. The next Figure 4.1 shows an example of realization where the first 15000 jumping times are computed. To compare the methods, we draw 15000 realizations of a random variable with exponential distribution and use this sequence for the 4 examples.

Note that we had to find a long enough trajectory because most ends quickly as the jumping time can be infinite. We see that for similar tolerances, our new method, called CHV, performs better as the number of jump events increases. Surprisingly, `ode45`, which is not designed for stiff systems performs quite well. We tried all available steppers in `Matlab` and only plot the two best results.

The following table 4.1 provides the running times (ratios) for the cases in Figure 4.1. Despite the additional dynamical variable τ of our method, the fastest running times using either methods are almost the same.

⁴for k fictitious jumps

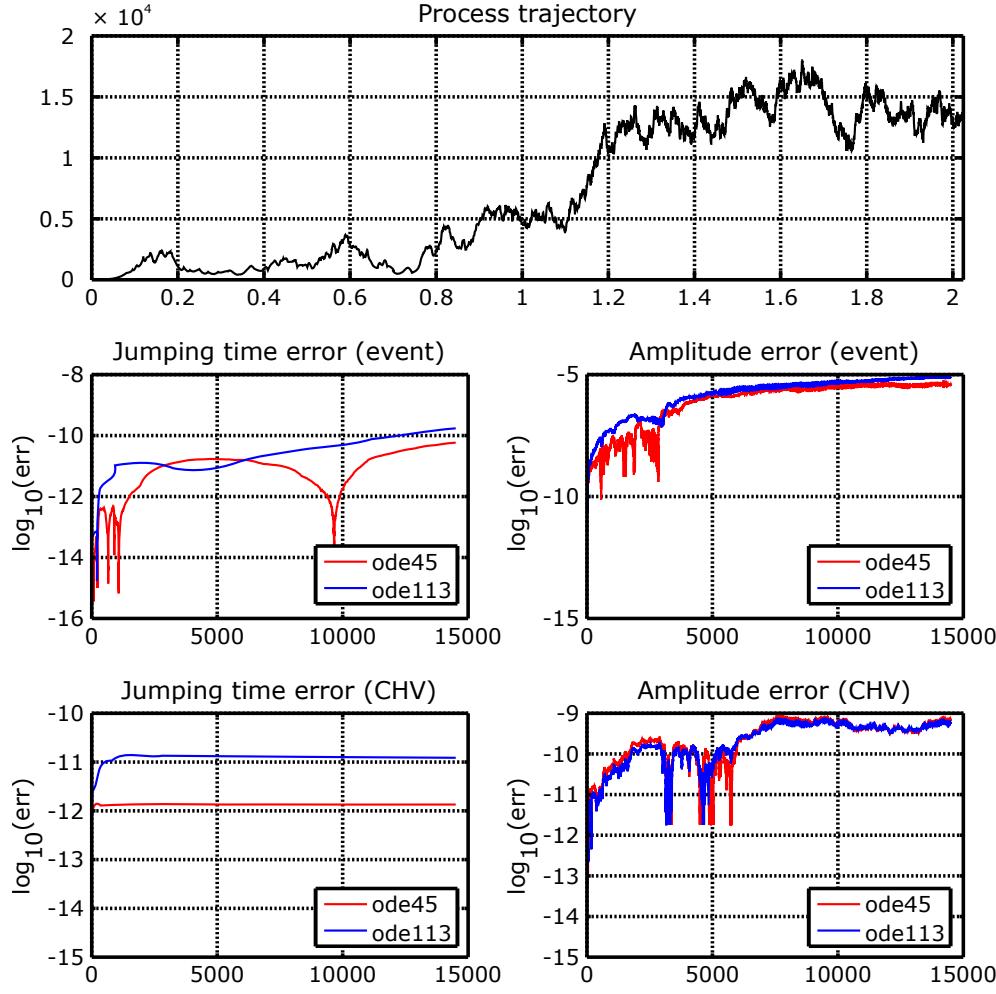


FIG. 4.1. Top: example of trajectory plotted using analytical formula (4.2). Bottom are errors in jumping time and amplitude of the process using the event location capabilities of `Matlab` (termed event here) and using our method (termed CHV). The horizontal axis is the jump index. In both cases, the 2 smallest errors were obtained using the steppers `ode113` and `ode45`. The absolute/relative tolerance are $\text{atol} = 1e-12$ and $\text{rtol} = 1e-12$.

	ode45	ode113
event	1	1.851
CHV	1.009	1.0967

TABLE 4.1

Computation time ratios for the examples in Figure 4.1.

4.2. Stiff problem with finite jumping times. We now provide a numerical example to show that the scheme in [Rie13] can lead to wrong dynamics. We chose the following process of switching dynamics where $X(t) = (x_c(t), x_d(t)) \in \mathbb{R}^2$:

$$\begin{cases} \dot{x}_c = 10 \cdot x_c & \text{if } x_d \text{ even} \\ \dot{x}_c = -3x_c^2 & \text{otherwise} \end{cases} \quad (4.3)$$

where the (unique) transition rate is $R_{tot}(x_c, x_d) = x_c$ associated with the jump $x_d \rightarrow x_d + 1$ for which $\Pi((x_c, x_d), dy) = \delta_{(x_c, x_d+1)}(dy)$. The jumping time solution of the integral equation reads (see Theorem 3.1):

$$\begin{bmatrix} X(T_n) \\ \tau(T_n) \end{bmatrix} = \begin{cases} \begin{bmatrix} 10 \cdot S_n + x_c(T_{n-1}) \\ \frac{1}{10} \log \left(1 + \frac{10S_n}{x(T_{n-1})} \right) \end{bmatrix} & \text{if } x_d \text{ even} \\ \begin{bmatrix} e^{-3S_n} x(T_{n-1}) \\ \frac{1}{3x(T_{n-1})} (e^{3S_n} - 1) \end{bmatrix} & \text{otherwise.} \end{cases} \quad (4.4)$$

We see that the function τ is well-defined meaning that the jumping times are always finite. The results are plotted in Figure 4.2. To compare the methods, we draw 20 realizations of a

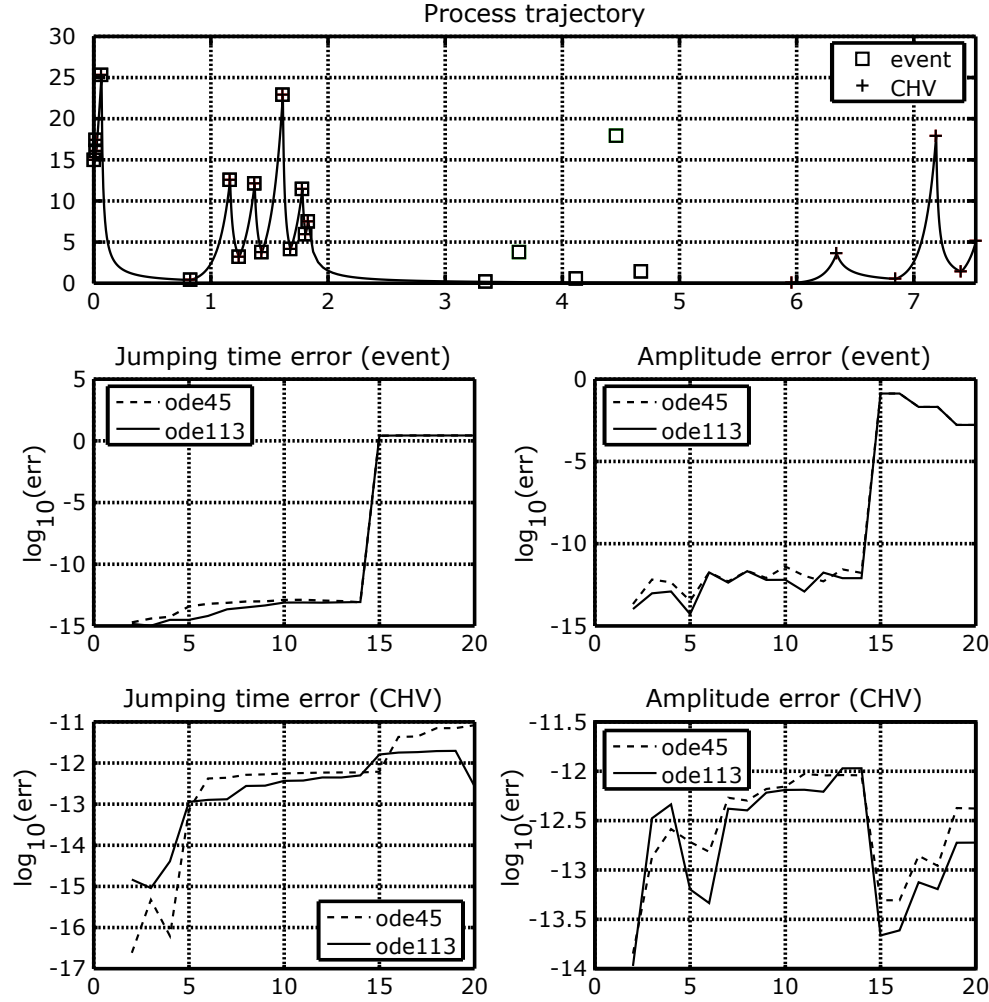


FIG. 4.2. Top: example of trajectory plotted using analytical formula (4.4). The green square are the jumps computed with the event function, the red crosses with our method. Bottom are errors in jumping time and amplitude of the process using the event location capabilities of `Matlab` (termed event here) and using our method (termed CHV). The horizontal axis is the jump index. In both cases, the 2 smallest errors were obtained using the steppers `ode113` and `ode45`. The absolute/relative tolerance are $atol = 1e - 12$ and $rtol = 1e - 12$.

random variable of exponential distribution and use this sequence for the 4 examples. After

a few jumps (on the order of ten), the method using the `Event` function in `Matlab` fails to compute accurately a jumping time when x_c is close to zero. This difficulty in computing the jumping time is also reflected in the cpu time shown in Table 4.2. Hence, computing 100 jumps with the Event method leads to completely wrong dynamics whereas our new method gives very satisfactory results when compared to the analytical results.

	ode45	ode113
event	4.941	3.569
CHV	1.940	1

TABLE 4.2

Computation time ratios for the examples in Figure 4.2.

4.3. Comparison with the rejection method. Finally, we provide a numerical example to compare our method to the FJM. We chose the following process of switching dynamics:

$$\begin{cases} \dot{x}_c = 3 \cdot x_c & \text{if } x_d \text{ even} \\ \dot{x}_c = -4x_c & \text{otherwise} \\ x_c(0) = 0.05, x_d(0) = 0 \end{cases} \quad (4.5)$$

where the (unique) transition rate is $R_{tot}(x_c, x_d) = \frac{1}{1.0 + e^{-x_c + 5.0}} + 0.1$ with $\Pi((x_c, x_d), dy) = \delta_{(x_c, x_d+1)}(dy)$. When the continuous component x_c is increasing, the total rate R_{tot} is bounded by 1. However, when it is decreasing, the total rate is bounded by $R_{tot}(x_c(T_n), x_d(T_n))$ for $t \geq T_n$ where T_n is the last jumping time. Hence, we expect the FJM to perform less fictitious jump when x_d is odd. The fact that x_c grows exponentially disfavors the FJM as we shall see.

To compare the two methods CHV and FJM, we use the adaptive ODE solver CVODE from SUNDIALS [HBG⁺05] for its ability to automatically switch between stiff and non-stiff methods. In each of the 4 cases, we compute 1 jump instant (resp. 3 jump instants) with each method and we perform 10^5 realizations. We show the results in Figure 4.3 where each plot is an histogram of the computing time in seconds of the jump instants. A first interesting result concerns the case *FJM - 1 jump* where it seems that we can see, in the distribution, the fictitious jumps used to compute the first jump. Indeed, in our implementation, it takes roughly $1.5ms$ to solve the ODE with both methods. The top two plots show that the CHV method is faster for the reasons stated at the beginning of the section. The other 2 plots show that this result is not affected by the computation of the jumping time in the case x_d odd, where the FJM should be slightly faster. Hence, the fastest method for simulating the process (4.5) would be to use our method when x_d is even and the rejection method otherwise.

5. Generalizations. Several possible generalizations of our model suggest themselves. Basically, these entail a modification of the flow (2.2) or of the jump process.

If the flow can be written as a Cauchy problem, then our method still applies albeit possibly in a different state space. For example, we can consider the case of a flow generated by a delay differential equation [Hal93] or if jump process depends on the history of the process $(X(t))$. These cases are handled very similarly to the case presented here, except that (2.2), (3.1) are now delay differential equations. One could also look at the case of a flow generated by partial differential differential equations [BR11].

The next possible generalization was introduced in [BVTH05] where a time delay was added between the initiation and the completion of some, or all, of the reactions. These delays were motivated by oscillations observed in gene regulation networks [DLD01]. A modified True Jump Method was given [BVTH05] where a table is added to record when the delayed

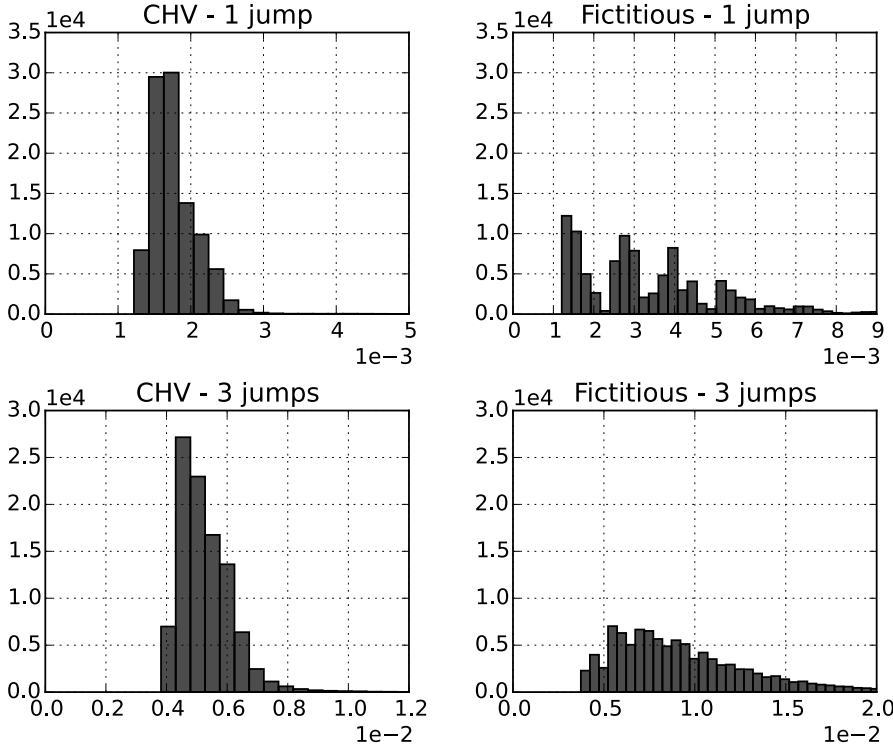


FIG. 4.3. Histograms of the computing time in seconds for each method CHV or FJM when computing 1 jump (top) and 3 jumps (bottom). The absolute/relative tolerance are $atol = 1e - 10$ and $rtol = 1e - 10$. See text for a description of the stochastic process.

reactions are scheduled. The algorithm was later improved by [Cai07, And07] but the method is the same. Hence, the computation of the next reaction time follows from the same integral equation that we solved here and our results equally apply.

6. Discussion. We have identified a new way to simulate a large class of Markov processes, the PDMPs, with a method which is versatile enough to be applied to various continuous part dynamics (ODE / PDE / delay differential equations) and any continuous rate function R_{tot} . Indeed the precision of the method is directly linked to the one of the ODE solver, which we assume is arbitrary given the state of the art concerning global error estimation [Ske86, Vis01].

As a particular case, our method provides a solution to cope with the notoriously difficult problem [And07] of the simulation of chemical reactions with time dependent propensity functions. Nevertheless, this problem is only formal as only the Fictitious Jump Method is exact in this case.

Our result is an important achievement for the following reasons. First, there is no need to modify existing ODE solvers to implement our method. Second, it provides a way to use an adaptive discretization to approximate *both* the ODE flow and the integral equation. As such, it makes the method [ACT⁺05, Rie13] for the simulation of PDMPs not relevant. It also allows to focus on the choice of the ODE solver without *ad-hoc* modification of existing tools, this is especially useful when studying stiff systems such as the slow-fast ones, largely present in the modeling of biology. Our results also makes it easy to consider flows driven by PDE or delay differential equations for example, without sacrificing for the numerical preci-

sion. Further, our results bridge the gap between the numerical methods for the simulation of PDMPs and the ones of ODE, making the first a subproblem of the second class. In particular, numerical error analysis [HW96, Ske86, Vis01, CP04] of ODE directly maps onto PDMPs.

The only relevant question which remains is whether to chose our method or the FJM when the continuous flow is not known analytically and when the total rate function R_{tot} is continuous. This is strongly dependent on the total rate function variations and not so much on its absolute values. If the probability of having at least 2 fictitious jumps is 'high', then our new method might be the most efficient while being simpler to program. Finally, note that mixing the two methods can be effective by choosing the fastest one depending on the value of the discrete component as we suggested in our third example.

Acknowledgment. This work was partially supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 269921 (BrainScaleS), no. 318723 (MathemacS), and by the ERC advanced grant NerVi no. 227747 and by the Human Brain Project (HBP).

Appendix A. True Jump Method. We recall the true jump method [GT13] for the exact simulation of $(X(t))_{t \in \mathbb{R}^+}$ on a time interval $[0, T]$. This is the SSA Doob-Gillespie algorithm [Gil77] when $F = 0$, for which $\phi^t = Id$.

Algorithm 1. (True Jump Method)

1. Draw the initial value X_0
2. Draw S_n according to an exponential distribution $\mathcal{E}(1)$
3. Set $T_n = \inf \left\{ u > T_{n-1} : \int_{T_{n-1}}^u R_{tot}(\phi^s(X(T_{n-1}))) ds = S_n \right\}$ and $X(t) = \phi^t(X(T_{n-1}))$ for $T_{n-1} \leq t < T_n$.
4. Draw $X(T_n)$ according to the law $\Pi(\phi^{T_n - T_{n-1}}(X(T_{n-1})), dy)$.
5. If $T_n < T$, return to step 2.

Appendix B. Fictitious Jump Method. We recall the "rejection" method [GT13] for the exact simulation of $(X(t))_{t \in \mathbb{R}^+}$ on a time interval $[0, T]$. We assume that $\sup_{x \in \mathbb{R}^d} R_{tot}(x) \leq \lambda < \infty$.

Algorithm 2. (Fictitious Jump Method)

1. Draw the initial value X_0
2. Draw S_n according to an exponential distribution $\mathcal{E}(\lambda)$
3. Set $X(t) = \phi^t(X(T_{n-1}))$ for $T_{n-1} \leq t < T_{n-1} + S_n$.
4. • either, with probability $1 - \frac{R_{tot}(X(T_{n-1}))}{\lambda}$, set $X(T_n) = X(T_{n-1})$
• or else draw $X(T_n)$ according to the law $\Pi(X(T_{n-1}), dy)$
5. If $T_n < T$, return to step 2.

REFERENCES

- [ACT⁺05] Aurélien Alfonsi, Eric Cancès, Gabriel Turinici, Barbara Di Ventura, and Wilhelm Huisinga. Adaptive simulation of hybrid stochastic and deterministic models for biochemical systems. In *ESAIM: Proceedings*, volume 14, pages 1–13. EDP Sciences, 2005.
- [And07] David F. Anderson. A modified next reaction method for simulating chemical systems with time dependent propensities and delays. *The Journal of Chemical Physics*, 127(21):214107, 2007.
- [Arn98] L. Arnold. *Random dynamical systems*. Springer, Berlin; New York, 1998.
- [BdSD10] Adrien Brandejsky, Benoîte de Saporta, and François Dufour. Numerical methods for the exit time of a piecewise-deterministic Markov process. *arXiv:1012.2659 [math]*, December 2010. arXiv: 1012.2659.
- [BR11] Evelyn Buckwar and Martin G. Riedler. An exact stochastic hybrid model of excitable membranes including spatio-temporal evolution. *Journal of Mathematical Biology*, 63(6):1051–1093, January 2011.

- [Bre14] Paul C. Bressloff. *Stochastic processes in cell biology*. Springer, 2014.
- [BVTH05] Dmitri Bratsun, Dmitri Volfson, Lev S. Tsimring, and Jeff Hasty. Delay-induced stochastic oscillations in gene regulation. *Proceedings of the National Academy of Sciences of the United States of America*, 102(41):14593–14598, 2005.
- [Cai07] Xiaodong Cai. Exact stochastic simulation of coupled chemical reactions with delays. *The Journal of Chemical Physics*, 126(12):124108, March 2007.
- [CP04] Yang Cao and Linda Petzold. A Posteriori Error Estimation and Global Error Control for Ordinary Differential Equations by the Adjoint Method. *SIAM Journal on Scientific Computing*, 26(2):359–374, January 2004.
- [CT97] Christiane Coccozza-Thivent. *Processus stochastiques et fiabilité des systèmes*. Springer Science & Business Media, September 1997.
- [Dav84] M. H. A. Davis. Piecewise-Deterministic Markov Processes: A General Class of Non-Diffusion Stochastic Models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 46(3):353–388, January 1984.
- [Dav93] M. H. A. Davis. *Markov models and optimization*. Monographs on statistics and applied probability. Chapman & Hall, London ; New York, 1st ed edition, 1993.
- [DLD01] Deanna L. Denault, Jennifer J. Loros, and Jay C. Dunlap. WC-2 mediates WC-1-FRQ interaction within the PAS protein-linked circadian feedback loop of *Neurospora*. *The EMBO Journal*, 20(1-2):109–117, January 2001.
- [GB00] Michael A. Gibson and Jehoshua Bruck. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.
- [Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
- [Gil01] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716, 2001.
- [GPB08] Bruno Gaujal, Florence Perronnin, and Remi Bertin. Perfect Simulation of a Class of Stochastic Hybrid Systems with an Application to Peer to Peer Systems. *Discrete Event Dynamic Systems*, 18(2):211–240, June 2008.
- [GT13] Carl Graham and Denis Talay. *Stochastic Simulation and Monte Carlo Methods*, volume 68 of *Stochastic Modelling and Applied Probability*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [Hal93] Jack K. Hale. *Introduction to Functional Differential Equations*. Springer, October 1993.
- [HBG⁺05] Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, September 2005.
- [Hin83] AC Hindmarsh. ODEPACK, A Systematized Collection of ODE Solvers, R. S. Stepleman et al. (eds.), North-Holland, Amsterdam, (vol. 1 of), pp. 55-64. *IMACS Transactions on Scientific Computation*, 1:55–64, 1983.
- [HW96] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II*, volume 14 of *Springer Series in Computational Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.
- [KK13] Hye-Won Kang and Thomas G. Kurtz. Separation of time-scales and model reduction for stochastic reaction networks. *The Annals of Applied Probability*, 23(2):529–583, April 2013. Zentralblatt MATH identifier 06157196, Mathematical Reviews number (MathSciNet) MR3059268.
- [PTW10] K. Pakdaman, M. Thieullen, and G. Wainrib. Fluid limit theorems for stochastic hybrid systems with application to neuron models. *Advances in Applied Probability*, 42(3):761–794, September 2010.
- [Rie13] Martin G. Riedler. Almost sure convergence of numerical approximations for Piecewise Deterministic Markov Processes. *Journal of Computational and Applied Mathematics*, 239:50–71, February 2013.
- [Sha03] Lawrence F Shampine. *Solving odes with matlab*. Cambridge Univ Press, New York, 2003.
- [Ske86] Robert D. Skeel. Thirteen ways to estimate global error. *Numerische Mathematik*, 48(1):1–20, 1986.
- [Vis01] Divakar Viswanath. Global errors of numerical ODE solvers and Lyapunov’s theory of stability. *IMA Journal of Numerical Analysis*, 21(1):387–406, January 2001.